SciSpot: Scientific Computing On Transient Cloud Servers

Paper # 105

ABSTRACT

Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers can be used to lower the costs of running applications on the cloud. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient cloud servers, and present SciSpot, a software framework for running scientific applications at low cost on the cloud. SciSpot's design is guided by our observation that most scientific computing applications (such as simulations) are deployed as "bag" of jobs, which represent multiple instantiations of the same computation with different physical and computational parameters. Treating bags of jobs as a unit of execution enables simple and powerful policies for optimizing the cost, makespan, and ease of deployment. SciSpot uses Google Cloud Preemptible VMs, and provides the first empirical and analytical model for their preemptions. SciSpot reduces costs by 5× compared to conventional cloud deployments, and makespans by up to 10× compared to conventional HPC clusters.

1 INTRODUCTION

Scientific computing applications play a critical role in understanding natural and synthetic phenomena associated with a wide range of material, biological, and engineering systems. The computational models and simulations for analyzing these systems can consume a large amount of computing resources, and require access to large dedicated high performance computing (HPC) infrastructure.

Increasingly, cloud computing platforms have begun to supplement and complement conventional HPC infrastructure to meet the large computing and storage requirements of scientific applications. Public cloud platforms such as Amazon's EC2, Google Cloud Platform, and Microsoft Azure, offer multiple benefits such as *on-demand* resource allocation, convenient pay-as-you-go pricing models, ease of provisioning and deployment, and near-instantaneous elastic scaling. Most cloud platforms offer *Infrastructure as a Service*, and provide computing resources in the form of *Virtual Machines (VMs)*, on which a wide range of applications such as web-services, distributed data processing, distributed machine learning, etc., are deployed.

To meet the diverse resource demands of different applications, public clouds offer resources (i.e., VMs) with multiple different resource configurations (such as number of CPU cores, memory capacity, etc.), and pricing and availability contracts. Conventionally, cloud VMs have been offered with "on-demand" availability, such that the lifetime of the VM is solely determined by the owner of the VM (i.e., the cloud customer). Increasingly however, cloud providers have begun offering VMs with *transient*, rather than continuous on-demand availability. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. Due to their volatile nature, transient VMs such as Amazon Spot instances, Google Preemptible VMs, and Azure Batch VMs, are offered at steeply discounted rates ranging from 50 to 90%.

While the on-demand resource provisioning and pay-as-you-go pricing makes it easy to spin-up computing clusters in the cloud, the deployment of applications on cloud platforms must be cognizant of the heterogeneity in VM sizes, pricing, and availability for effective resource utilization. Crucially, optimizing for *cost* in addition to makespan, becomes an important objective in cloud deployments. Furthermore, although using transient resources can drastically reduce computing costs, their preemptible nature results in frequent job failures. While preemptions can be mitigated with additional fault-tolerance mechanisms and policies [34, 38], these policies must be typically tailored to the application [38], and impose additional performance and deployment overheads. These considerations of cost, server configuration heterogeneity, and frequent job failures intrinsic to the system present multiple challenges in deploying applications on cloud platforms which are *fundamentally* different from those that appear in using HPC clusters as the execution environment for the scientific computing applications.

In this paper, we develop principled approaches for deploying and orchestrating scientific computing applications on the cloud, and present SciSpot, a framework for low-cost scientific computing on transient cloud servers.

Our policies for tackling the resource heterogeneity and transient availability of cloud VMs build on a key insight: most scientific computing applications are deployed as a collection or "bag" of jobs. These bags of jobs represent multiple instantiations of the same computation with different parameters. For instance, each job may be running a (parallel) simulation with a set of simulation input parameters, and different jobs in the collection run the same simulation employing a different set of parameters. Collectively, a bag of jobs can be used to "sweep" or search across a multi-dimensional parameter space to discover or narrow down the set of feasible and viable parameters associated with the modeled natural or synthetic processes. A similar approach is adopted in the use of machine learning (ML) to enhance scientific computational methods, a rapidly emerging area of research, when a collection of jobs with independent parameter sets are launched to train ML models to predict simulation results and/or accelerate the simulation technique.

, ,
.

Prior approaches and systems for mitigating transiency and cloud heterogeneity have largely targeted individual instantiations of jobs [34, 38, 39, 43]. For a bag of jobs, it is not necessary, or sufficient, to execute an individual job in timely manner—instead, we could selectively restart failed jobs in order to complete the necessary, desired subset of jobs in a bag. Furthermore, treating the bag of jobs as a fundamental unit of computation allows us to select the "best" server configuration for a given application, by exploring different servers for initial jobs and running the remainder of the jobs on the optimal server configuration.

We show that optimizing across an entire bag of jobs and being cognizant of the relation between different jobs in a bag, can enable simple and powerful policies for optimizing cost, makespan, and ease of deployment. We implement these policies as part of the SciSpot framework, and make the following contributions:

(1) In order to select the "right" VM from the plethora of choices offered by cloud providers, we develop a cluster configuration policy that minimizes the cost of running applications. Our search based policy selects a transient server type based on it's cost, parallel speedup, and probability of preemption.
(2) Since transient server preemptions can disrupt the execution of jobs, we present the *first* empirical model and analysis of transient server availability that is *not* rooted in classical bidding models for EC2 spot instances that have been proposed thus far. Our empirical model allows us to predict expected running and costs of jobs of different types and duration.
(3) We develop preemption-mitigation policies to minimize the overall makespan of bags of jobs, by taking into consideration the partial redundancy between different jobs within a bag. Combined, our policies yield a cost saving of 70%, and a makespan reduction of 20% compared to a conventional HPC clusters.

2 BACKGROUND AND OVERVIEW

In this section, we give on overview of the characteristics and challenges of transient cloud computing; motivate the need for the bag of jobs abstraction in scientific computing workflows; and give an overview of our SciSpot system.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an "on-demand" basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variations. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [**? ?**].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers with *transient* availability, which can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [**?**], Google Preemptible VMs [**?**], and Azure batch VMs [**?**], have become popular in recent years due to their discounted prices, which can be 7-10x lower than conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as Packet [**?**] and Alibaba [**?**] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [40**?**]. Preemptions are akin to fail-stop failures, and result in loss of the application's memory and disk state, leading to downtimes for interactive applications such as web services, and poor throughput for long-running batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [34, 38, 43] and resource management techniques [39] to ameliorate the effects of preemptions for a wide range of applications. However, the effect of preemptions is dependent on a combination of application resource and fault model, and mitigating preemptions for different applications remains an active research area [24].

2.2 Bag of Jobs in Scientific Computing

The typical workflow associated with most scientific computing applications, often involves evaluating a computational model across a wide range of physical and computational parameters. For instance, constructing and calibrating a molecular dynamics application (such as [25]), usually involves running a simulation with different physical parameters such as characteristic sizes and interaction potentials, as well as computational parameters such as simulation timesteps. Each of these parameters can take a wide range of values, resulting in a large number of combinations which must be evaluated by invoking the application mulitple times (also known as a parameter sweep). Since each computational job explores a single combination of parameters, this results in executing a "bag of jobs", with each job in the bag running the same application, but with possibly different parameters.

The bag of jobs execution model is pervasive in scientific computing and applicable in many contexts. In addition to exploratory parameter sweeps, bags of jobs also result from running the application a large number of times to account for model or computational stochasticity, and can be used to obtain tighter confidence intervals. Increasingly, bags of jobs also arise in the emerging research that combines statistical machine learning (ML) techniques and scientific simulations [3, 6, 10, 11, 25, 26, 32, 33, 37, 47]. For instance, large bags of jobs are run to provide the necessary training and testing data for learning statistical models such as neural networks that are then used to improve the efficacy of the simulations.

The bag of jobs execution model has multiple characteristics, that give rise to unique challenges and opportunities when deploying them on cloud transient servers. First, since bags of jobs require a large amount of computing resources, deploying them on the cloud can result in high overall costs, thus requiring policies for minimizing the cost and overall running time. Second, we observe that usually, there is no depedency between individual jobs in a bag, thus allowing increased flexibility in job scheduling. And last, treating entire bags of jobs as an execution unit, instead of individual jobs, can allow us to use partial redundancy between jobs and reduce the fault-tolerance overhead to mitigate transient server preemptions.

2.3 SciSpot Overview

Our system, SciSpot, is a general-purpose software framework for running scientific computing applications on low-cost cloud transient servers. It incorporates policies and mechanisms for generating, deploying, orchestrating, and monitoring bags of jobs on cloud servers. Specifically, it runs a bag of jobs defined by these parameters:

Bag of job = {$\mathcal{A}$: Application to execute,
N: Number of jobs,
m: Minimum number of jobs to finish,
π: Generator function for job parameters,
$\mathcal{R}$: Computing resources per job}

SciSpot seeks to minimize the cost and running time of bags of jobs of scientific computing applications. SciSpot's cost and time minimizing policies for running bags of jobs are based on empirical and analytical models of the cost and preemption dynamics of cloud transient servers, which we present in the next section.

3 PREEMPTION DYNAMICS OF TRANSIENT CLOUD SERVERS

To measure and improve the performance of applications running on transient cloud servers, it is critical to understand the nature and dynamics of their preemptions. The preemption characteristics are governed by the supply of surplus resources, the demand for cloud resources, and the resource allocation policies enforced by the cloud operator. In this section, we present empirical and analytical models that describe these characteristics and enable an intuitive understanding of the nature of preemptions.

3.1 The need for empirical preemption models

Amazon's EC2 spot instances were the original cloud transient servers. The preemptions of EC2 spot instances are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server is preempted.

Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" of the spot instances. Many research projects have used

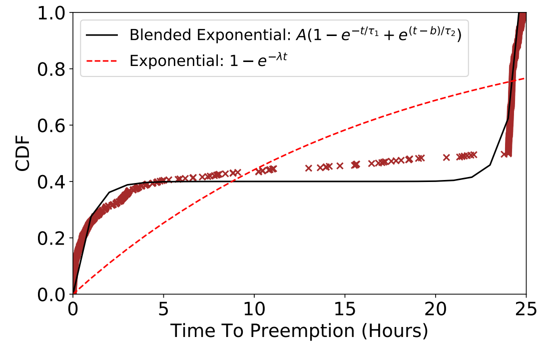

Figure 1: CDF of lifetimes of Google Preemptible Instances. Our blended exponential distribution fits much better than the conventional exponential failure distributions.

publicly available[1] historical spot prices to characterize and model spot instance preemptions [40**?**]. For example, past work has analyzed spot prices and shown that the MTTF's of spot instances of different hardware configurations and geographical zones ranges from a few hours to a few days [**?** **?**].

However, Amazon has recently changed the preemption characteristics of spot instances, and servers are now preempted even if the spot price is below the maximum price. Thus, spot prices are no longer a completely reliable indicator of preemptions, and preemptions can no longer be inferred from looking at prices alone. Therefore, new techniques are required to model preemption dynamics that can supplement the earlier price-based approaches, and we develop these techniques next.

3.2 Empirical preemption behavior

The preemptions of transient servers need not be related to their price. For example, Google's Preemptible VMs and Azure Batch VMs have a *fixed* price relative to their non-preemptible counterparts. In such cases, price based models are inadequate, and other approaches to understand preemptions are required.

This task is further complicated by the fact that these cloud operators (Google and Microsoft) do not currently provide any information about preemption characteristics. Thus, relatively little is known about the preemptions (and hence the performance) of these transient VMs.

In order to understand preemption dynamics of transient servers, we conduct a large-scale empirical measurement study which is the first of its kind. We launched more than 1000 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (aka, their useful lifetime).[2]

[1] Amazon posts Spot prices of 3 months, and researchers have been collecting these prices since 2010 [**?**].
[2] We will release the complete preemption dataset and hope that other researchers can benefit.

A sample of 100 such preemption events are shown in Figure 1, which shows cumulative distribution of the VM lifetimes. Note that the cloud operator (Google) caps the *maximum* lifetime of the VM to 24 hours, and all the VMs are preempted before that hard limit. Furthermore, the lifetimes of VMs are *not* uniformly distributed, but have three distinct phases. In the first phase, characterized by VM lifetime $t \in (0, 3)$ hours, we observe that many VMs are quickly preempted after they are launched, and thus have a steep rate of failure initially; the rate of failure or preemptions can be obtained by taking the derivative of the CDF. The second phase characterizes the VMs that survive past 3 hours and enjoy a relatively low and uniform preemption rate over a relatively broad range of lifetime (characterized by the slowly rising CDF in Figure 1). The final phase exhibits a steep increase in the number of preemptions as the preemption deadline of 24 hours approaches. The overall rate of preemptions is "bath tub" shaped.

We note that this preemption behavior, imposed by the constraint of the small, 24 hour lifetime, is *substantially* different from conventional failure characteristics of hardware components and even EC2 spot instances. In these "classical" setups, the rate of failure usually follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. Figure 1 shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. From Figure 1, we can see that the classic exponential distribution is unable to model the observed preemption characteristics. We attribute this deficiency to the central assumption made in the underlying reliability theory principles that leads to the exponential distribution: the rate of preemptions is independent of the lifetime of the VMs, in other words, the preemptions are *memoryless*. This assumption breaks down when there is a fixed upper bound on the lifetime, as is the case for Google Preemptible VMs, and the conventional approach becomes insufficient to model this constrained preemption dynamics.

3.3 Analytical model of preemption dynamics in Google cloud

We now develop a *minimal* analytical model for preemption dynamics that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations presented in Section 4. This new model is based on the earlier observation that the cumulative distribution of lifetimes has multiple distinct temporal phases. The key assumption underlying our minimal model is the presence of two distinct failure processes that give rise to a new probability distribution characterizing the preemptions and the observed CDF, and ensure the dependence of the rate of failure on the VM lifetime. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the steep rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed to be characterized by an exponential term that captures the sharp rise in preemptions that results from the constraint of a fixed 24 hour lifetime. Generally, these two processes compete during the middle phase to yield a relatively constant and low number of preemptions; in practice, based on the fits to the empirical data, we observe the first process to dominate over the second during this phase as well.

We propose the following general form for the CDF based on this model:

$$\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right), \tag{1}$$

where $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase (generally, $1/\tau_2 > 1/\tau_1$), b denotes the time when the preemptions occur at a high rate (generally, around 24 hours) which we term the activation time for the second process, and A is a constant used to scale the CDF to ensure that the initial conditions ($F(0) = 0$) are met.

For most of its life, a VM sees failures according to the classic exponential distribution with a rate of failure equal to $1/\tau_1$ – this behavior is captured by $1 - e^{-t/\tau_1}$ term in Eq. 1. As VMs get closer to their maximum lifetime (24 hours) imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high, exponential rate, which is captured by the second term introduced in the CDF ($e^{(t-b)/\tau_2}$). Shifting the argument (t) of the exponential by b ensures that the exponential reclamation is only applicable towards the end of the VM's maximum lifetime and does not dominate over the entire temporal range. As noted before, $1/\tau_2$ is the rate of this reclamation.

The analytical model and the associated 4 parameter distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters characterizing the preemption dynamics can be obtained for a given empirical CDF by minimizing least-squared function fitting methods. [3] In the next section, we use this analytical model for optimizing cloud resource selection such that we can run scientific computing applications at low cost and running times. We note that our motivation here is to provide a minimal model, i.e. a model based on data-driven observations and reasonable assumptions that provides a sufficiently accurate description of constrained preemption dynamics with the minimal number of necessary parameters. As is evident from Figure. 1, the analytical $\mathscr{F}$ shows deviations from the data near the halfway point within the 24 hour lifetime. One can envision generalizing this model by including more failure processes characterized by failure rates and activation times (like b) to capture the data with higher accuracy. Of course, this introduces a higher number of parameters and reduces the predictive power and simplicity of the model.

[3] More details about the distribution fitting are presented in the implementation section(**??**

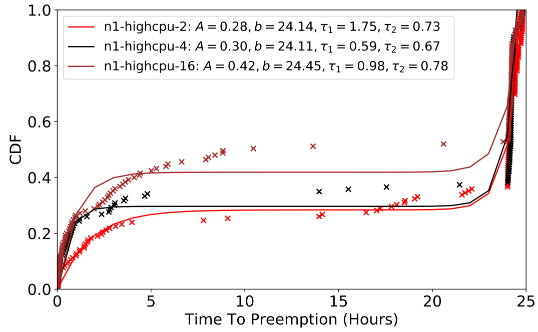

Figure 2: The preemption characteristics of different VM types. Larger VMs are more likely to be preempted

3.4 Preemption dynamics of VMs of different types

Since cloud platforms support a wide range of applications, they also offer a large range of servers (VMs) with different resource configurations (such as the number of CPU cores, memory size, I/O bandwidths, etc.). For example, a cloud provider may offer VMs with (4 CPUs, 4 GB memory), (8 CPUs, 8 GB memory), etc. Most clouds offer a large number of different hardware configurations—Amazon EC2 offers more than 50 hardware configurations, for example [?].

In general, the preemption dynamics of a VM are determined by the supply and demand of VMs of that *particular* type. Thus, the preemption characteristics of VMs of different sizes and running in different geographical zones are different. Figure 2 shows the preemption CDF's of three such VM types in the Google Cloud, along with the parameters of our four parameter distribution. We show the data from three different types of VMs `n1-highcpu-{2,8,32}`, where the number indicates the number of CPU's.

From Figure 2, we see that our distribution is able to capture the preemption dynamics of different VM types. Interestingly, we can also observe that larger VMs have a higher rate of failure. This is because larger VMs require more computational resources (such as CPU and memory), and when the supply of resources is low, the cloud operator can reclaim a large amount of resources by preempting larger VMs. This observed behavior aligns with the guidelines for using preemptible VMs that suggests the use of smaller VMs when possible [?].

Our analytical model also helps use crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM of type i, as

$$E[L_i] = \int_0^{24} t f_i(t)\ dt \qquad (2)$$

Where $f(t) = \dfrac{d\mathscr{F}(t)}{dt} = A\left(\dfrac{1}{\tau_1}e^{-t/\tau_1} + \dfrac{t-b}{\tau_2}e^{\frac{t-b}{\tau_2}}\right)$

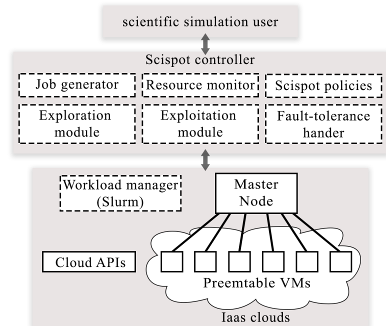

Figure 3: SciSpot Architecture.

Since preemptions require restarting a job and increase the job completion time, it may be more prudent to select transient VMs with higher expected lifetimes. We use the analytically derived expected lifetimes of VMs of different types in SciSpot when selecting the the "best" VM type for a given bag of jobs. This server selection is a key part of SciSpot design, which we describe next.

4 SCISPOT DESIGN

SciSpot is designed as a framework that increases the usability and viability of transient cloud servers for scientific computing applications, and provides a simple user interface to allow users to deploy their applications with minimum workflow changes. Most scientific computing applications are deployed on HPC clusters that have a batch scheduler such as Slurm [?] or Torque [?], and SciSpot integrates with these schedulers (e.g., Slurm) to provide the same interface to applications. As shown in Figure 3, SciSpot creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the various policies described in the rest of this section.

High-level workflow: When a user wishes to run a bag of jobs, SciSpot handles the provisioning of a cluster of transient cloud servers. In addition, SciSpot deals with the scheduling and monitoring of the bag of jobs, and with VM preemptions. Execution of a bag of jobs proceeds in two phases. In the first phase, SciSpot selects the "right" cluster configuration for a given application through a cost-minimizing exploration-based search policy, described in Section 4.1. In the second phase, SciSpot proceeds to run the remaining jobs in the bag on the optimal cluster configuration.

4.1 Server Selection

4.1.1 Why Server Selection is Necessary. Before deploying any application on the transient cloud servers, we must first select the appropriate cloud server for the application. Since cloud platforms support a wide range of applications, they also offer a large range of servers (VMs) with different resource configurations (such as the number of CPU cores, memory size, I/O bandwidths, etc.). For example, a cloud provider may offer VMs with (4 CPUs, 4 GB memory), (8 CPUs, 8 GB memory), etc. Most clouds offer a large number of different hardware configurations—Amazon EC2 offers more than 50 hardware configurations, for example [?]. *Importantly, different server configurations have different cost, performance, and preemption characteristics.*

Even if we assume that the total amount of resources to be allocated to a job is fixed, there are multiple *cluster configurations* to satisfy the allocation with the large number of available server types. Server selection is especially important for parallel applications, because although the total amount of resources in each cluster configuration is constant, the resources are distributed differently—i.e., a job can run on either 2 VMs with 32 CPUs each, or a single 64 CPU VM. Since the performance of parallel applications is particularly sensitive to their communication overheads, different cluster configurations may yield different job running times. For instance, a smaller cluster with large VMs will result in lower inter-VM communication, and thus shorter running times.

However, the performance of an application is also affected by preemptions of transient servers. Since preemptions are essentially fail-stop failures, synchronous parallel applications (such as those using MPI) are are forced to abort, and completing the job requires restarting it. Thus, frequent preemptions can increase the overall turnaround time of a job.

4.1.2 Server Selection Policy. Having provided the motivation and tradeoffs in server selection, we now describe the SciSpot's server selection policy. Given an application and a bag of jobs, SciSpot "explores" and searches for the right server type by minimizing the expected cost of running the job. Since jobs in a bag have similar execution characteristics, optimizing server selection for an individual job also translates to the entire bag.

SciSpot allows the users to specify the total amount of resources required per job, which we denote by $\mathcal{R}$. For example, $\mathcal{R}$ can be the total number of CPU cores. We first determine the search space, which is the space of all cluster configurations (i, n_i) such that $r_i n_i = \mathcal{R}$, where r_i is the resource size of a VM of type i (e.g., number of CPUs), and n_i is the number of VMs of that type. Based on the constraint, the number of servers of type i required, $n_i = \mathcal{R}/r_i$.

Each cluster configuration yields different application performance, preemption overhead, and cost. Our aim is to find the lowest-cost configuration (i, n_i) for a given application, which we do through an exploratory search. Our server selection policy runs the application on different cluster configurations to determine the base running time (in the absence of preemptions), which is denoted by $T_{(i,n_i)}$. It then combines the empirical running time with a cost model, to estimate the expected cost of running the application.

4.1.3 Server Cost Model. Since server selection involves a tradeoff between cost, performance, and preemptions, we develop a model that allows us to optimize the resource allocation and pick the best VM that minimizes the expected cost of running an application on SciSpot.

Let us assume that the cloud provider offers N server types, with the price (per unit time) of a server type equal to c_i. The overall expected cost of running a job can then be expressed as follows:

$$E[C_{(i,n_i)}] = n_i \times c_i \times E[\mathcal{T}_{(i,n_i)}] \tag{3}$$

Here, $E[\mathcal{T}_{(i,n_i)}]$ denotes the expected turnaround time of the job (accounting for preemptions) on n_i servers of type i.

This turaround time depends on the preemption probability of the server type, and can be expressed as:

$$E[\mathcal{T}_{(i,n_i)}] = T_{(i,n_i)} + E[\text{Recomputation Time}] \tag{4}$$

Here, $T_{(i,n_i)}$ is the base running time of a job without preemptions, which we obtain empirically as explained in the previous subsection. Since jobs have to be rerun when they fail due to preemptions, we define the recomputation time as:

$$E[\text{Recomputation Time}] = \frac{1}{2} * P(\text{at least one preemption}) * T_{(i,n_i)} \tag{5}$$

Our expression of the recomputation time assumes jobs will fail at the half-way mark on average, as is frequently assumed and observed [5, 8].

The probability that at least one VM out of n_i will be preempted during the job execution can be expressed as:

$$P(\text{at least one preemption}) = 1 - P(\text{no preemptions}) \tag{6}$$
$$= 1 - (1 - P(i,t))^{n_i} \tag{7}$$

Here, $P(i,t)$ denotes the probability of a preemption of a VM of type i when a job of duration t runs on it. It depends on the type of server, and its associated expected lifetime, and is defined as:

$$P(i,t) = \min\left(\frac{t}{E[L_i]}, 1\right) \tag{8}$$

Where $E[L_i]$ is the expected lifetime of the VM of type i extracted using the preemption model (Equation 2). We also assume that the running time of *individual* jobs in a bag (T), will be smaller than the expected lifetime of the VMs, otherwise we will see no forward progress since the jobs will always be preempted before completion. This is a safe assumption, since more than 90% HPC jobs are less than 2 hours long (Figure 7), and the average expected lifetime of transient VMs is ten hours or more. Note that this restriction only applies to individual jobs—SciSpot can smoothly run large bags of jobs even if their total running time exceeds the VM lifetime.

Using Equations 4,5, and 6, the overall expected cost of running a job on transient cloud servers is obtained as

$$E[C_{(i,n_i)}] = \frac{1}{2} n_i c_i T_{(i,n_i)} \left(3 - \left(1 - \frac{T_{(i,n_i)}}{E[L_i]}\right)^{n_i}\right) \tag{9}$$

Equation 9 shows that the expected cost $E[C]$ is higher for larger number of servers (high n_i), while it is reduced if the expected lifetime of the VM is larger (high $E[L_i]$). Thus, if we select VMs of smaller size, we will require more of them (higher n_i), and this cluster configuration will have a larger probability of failure and thus higher running times and costs. However, there is a tradeoff: selecting larger VMs results in smaller n_i, but larger VMs have higher preemption probability, as we have seen in Section 3.4.

To limit the search space, we observe that since most scientific computing applications are CPU bound, we only need to consider VMs meant for CPU-bound workloads, such as `highcpu` VMs in Google Cloud and the `cc` family in Amazon EC2. For example, the Google cloud offers a total of 7 `highcpu` server types with 1, 2, 4, 8, 16, 32, and 64 CPU's—yielding a small upper bound on the number of configurations to search. Furthermore, a large cluster of small servers is suboptimal for most applications (except those that are completely embarassingly parallel and have no communication). SciSpot thus explores VM's in descending order of their size and ignores exploring the small VMs (with 2 CPUs or fewer)—reducing the search space even further.

4.2 Scheduling a Bag of Jobs

Once the right cluster configuration for a job has been determined, SciSpot then proceeds to run the remaining jobs in the bag. A bag of jobs is determined by the total number of jobs in the bag, associated parameters for each job, and the minimum number of jobs that must be successfully executed. Given these parameters as input, SciSpot then creates a cluster by launching preemptible VMs and starts scheduling the different jobs in a bag. Upon job completion, we launch the next job in the bag and run it on the existing cluster of preemptible VMs. This policy is based on our preemption dynamics model which shows that preemptible rates have a "bath-tub" shape. Thus, jobs launched on "stable" VMs that have been running for a few hours, face low likelihood of failures—thereby reducing the number of job failures for the entire bag.

SciSpot also allows users to specify a deadline for bag completion, which we use to compute the number of jobs to execute in parallel. If the deadline specified is D, then the number of parallel jobs is $k = D/E[\mathcal{T}]$. Thus if the exploration phase recommends n_i VMs, then we launch a cluster of $k \times n_i$ VMs, with each job executing on n_i VMs. For this calculation, we assume that the running time of different jobs in a bag will largely be similar, but this is not a correctness requirement. Thus because of the stochasticity in job running times and VM lifetimes, SciSpot only meets the deadline in a "best effort" manner, and does not guarantee strict makespan constraints.

Upon job completion, the next job in the bag is run. When a job fails due to VM preemption, SciSpot replenishes the cluster by launching replacement VM's and resubmits the job. Jobs are restarted from a checkpoint if available. We do not restart failed jobs as long as we can complete the minimum number of jobs in the bag. Due to high demand, preemptible VM's of the chosen type may not be available. In such cases, SciSpot runs in a "degraded" mode—jobs are either run on a smaller number of VMs, or are run on VMs of a different size that are available but may have suboptimal cost.

Checkpointing Policy. When applicable, SciSpot resumes jobs from the latest checkpoint performed using tools such as DMTCP [**?**]. Since checkpointing also increases the running time of the job, the checkpoint interval must be carefully computed. The classic Young-Daly periodic checkpointing interval [8] is only applicable when failures follow an exponential distribution, which we have shown not to be true in the case of Google Preemptible VMs (Figure 1). Our analytical model for preemptions permits advanced, non-periodic checkpointing intervals that can be computed using a dynamic programming approach similar to [5].

5 SCISPOT IMPLEMENTATION

SciSpot is implemented as a light-weight, extensible framework that makes it convenient and cheap to run scientific computing applications in the cloud. We have implemented the SciSpot prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [**?**].

SciSpot is implemented as a centralized controller, which implements the server selection and job scheduling policies described in Section 4. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit bags of jobs to the controller via the HTTP API, and the controller then launches and maintains a cluster of cloud VMs, and maintains status of each job in a local json database. As a convenience feature, SciSpot also can also automatically generate parameter combinations for a given bag size—based on a user-provided json file that provides start and end values for each parameter.

SciSpot integrates, and interfaces with two primary services. First, it uses the Google cloud API [**?**] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm or Torque, to which it submits jobs. The current SciSpot prototype supports the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. SciSpot monitors job completions and failures (due to VM preemptions) through the use of slurm call-backs, which issue HTTP requests back to the slurm controller.

As part of SciSpot, we also provide a base VM image with Slurm and MPI integration, along with commonly used libraries and benchmarks for scientific computing. To run an application, users must provide a location to the application source code or binaries. Integrating SciSpot with container-based image management tools such as Docker and Singularity is currently part of our ongoing work.

6 EXPERIMENTAL EVALUATION

In this section, we present empirical and analytical evaluation of the performance and cost of SciSpot under different workloads and scales. Our evaluation consists of empirical analysis of the different scientific computing applications, as well as model-driven simulations for analyzing and comparing SciSpot behavior under different preemption and application dynamics.

Environment and Workloads: All our empirical evaluation is conducted on the Google Public Cloud, and with these representative scientific computing applications:

Nanoconfinement. The nanoconfiment application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [20, 27].
Shapes. The Shapes application runs an MD-based optimization dynamics to predict the optimal shape of deformable, charged nanoparticles [19, 23].
LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) code is a popular code to for hydrodynamics simulations of continuum material models [28, 29].

These examples are representative of typical scientific computing applications in the broad domain of physics, materials science, and chemical engineering. These three examples are implemented as parallel programs that use OpenMP and MPI parallel computing techniques. The first two are used in nanoscale materials research [17–20, 23, 41] and the LULESH is a widely used benchmark [28, 29]. All applications are run with default parameters unless otherwise stated. All applications use OpenMPI, are deployed on Slurm vXXX and 64-bit Ubuntu 18.04, and run on Google Cloud VMs with x86-64 Intel Broadwell CPUs.

6.1 SciSpot Performance and Cost

6.1.1 Impact of server exploration. As described in Section 4, applications can be deployed on multiple types of VMs in the cloud, with each VM type having a different "size". In our evaluation of parallel scientific computing applications that are CPU intensive, we are primarily interested in the number of CPUs in a VM.

When an application (i.e., bag of jobs) requests a total number of CPUs to run each of its jobs, SciSpot first runs its exploration phase to find the "right" VM for the application. SciSpot searches for the VM that minimizes the total expected cost $E[C_{(i,n_i)}]$ of running the application, and this depends on several factors such as the parallel structure of the application, the preemption probability and the associated job recomputation time, and the price of the VM.

Thus, even if the *total* amount of resources (i.e., number of CPUs) per job is held constant, the total running time (i.e., turnaround time) of an application depends on the choice of the VM type (i), and the associated number of VMs (n_i) required to meet the allocation constraint (Section 4.1.3). With preemptible instances, the total running time of a job is composed of two factors: the "base" running time of the job without any preemptions ($T_{(i,n_i)}$), and the expected recomputation time which depends on the probability of job failure (Equation 5).

Figure 4 shows the running times of the Nanoconfinement and Shapes application, when they are deployed on different VM sizes. In all cases, the total number of CPUs per job is set to 64, and thus the different VM sizes yield different cluster sizes (e.g., 16 VMs with 4 CPUs or 32 VMs with 2 CPUs).

For the Nanoconfinement application, we observe that the base running times (without preemptions) reduce when moving to larger VMs, because this entails lower communication costs. The running time on the "best" VM (i.e., with 32 CPUs) is nearly 40% lower as compared to the worst case. On the other hand, the Shapes application can scale to a larger number of VMs without any significant communication overheads, and does not see any significant change in its running time.

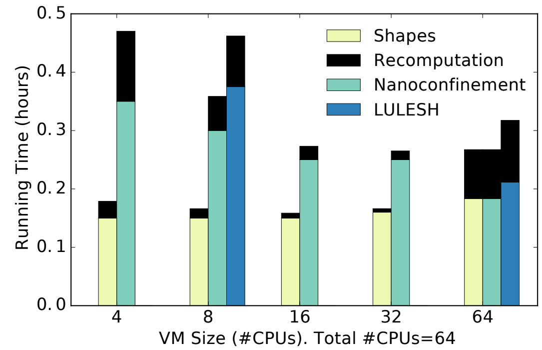

Figure 4: Running times of applications on different VMs. The total number of CPUs is 64, yielding different number of VMs in each case. We see different tradeoffs in the base running times and recomputation costs for the different applications.

Figure 4 also shows the expected turnaround time $E[\mathcal{T}_{(i,n_i)}]$, that is obtained by adding the the expected recomputation time, which depends on the expected lifetimes of the VM and the number of VMs, and is computed using the cost model introduced in Section 4.1.3. While selecting larger VMs may reduce communication overheads and thus improve performance, it is not an adequate policy in the case of preemptible VMs, since the preemptions can significantly increase the turnaround time. We can observe this in the case of Nanoconfinement application when deployed on a 64 CPU VM—even though the base running time is lower compared to deploying the application on 2x32-CPU VMs, the recomputation time on the 64 CPU VM is almost 4× higher due to the much lower expected lifetime of the larger VMs. Thus, on preemptible servers, there is a tradeoff between the base running time which only considers parallelization overheads, and the recomputation time. By considering *both* these factors, SciSpot's server selection policy can select the best VM for an application.

Result: *SciSpot's server selection, by considering both the base running time and recomputation time, can improve performance by up to 40% , and can keep the increase in running time due to recomputation to less than 5%.*

6.1.2 Cost. The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. Figure 5 compares the cost of running different applications with different cloud VM deployments. SciSpot, which uses both cost-minimizing server selection, and preemptible VMs, results in significantly lower costs across the board, even when accounting for preemptions and recomputations. Even with SciSpot's server selection, using on-demand VMs result in a 5× cost increase compared to SciSpot. In the absence

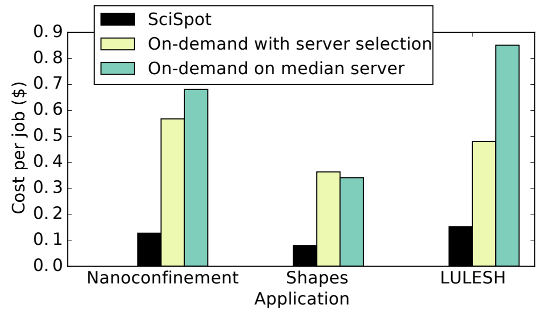

Figure 5: SciSpot's use of preemptible VMs can reduce costs by up to 5× compared to conventional cloud deployments.

of server selection, we assume that the user will pick a "median" VM in terms of number of CPUs (in this case, 8 CPU VMs), which we also show in Figure 5. Note that since SciSpot's server selection considers the turnaround time (which includes recomputation time), it may not always pick the optimal on-demand server.

Result: *SciSpot reduces computing costs by up to 5× compared to conventional on-demand cloud deployments.*

6.1.3 Comparison with HPC Overhead. Scientific applications are typically run on large-scale HPC clusters, where different performance and cost dynamics apply. While there are hardware differences between cloud VMs and HPC clusters that can contribute to performance differences, we are interested in the performance "overheads". In the case of SciSpot, the job failures and recomputations increase the job turnaround time, and are thus the main source of overhead.

On HPC clusters, jobs enjoy significantly lower recomputation probability, since the hardware on these clusters has MTTFs in the range of years to centuries [**?**]. However, we emphasize that there exist *other* sources of performance overheads in HPC clusters. In particular, since HPC clusters have high resource utilization, they also have significant *waiting* times. On the other hand, cloud resource utilization is low [**?**] and there is usually no need to wait for resources, which is why transient servers exist in the first place.

Thus, we compare the performance overhead due to preemptions for SciSpot, and job waiting times in conventional HPC deployments. To obtain the job waiting times in HPC clusters, we use the LANL Mustang traces published as part of the Atlas trace repository [2]. We analyze the waiting time of over two million jobs submitted over a 5 year period, and compute the increase in running time of the job due to the job waiting or queuing time.

We define the overhead as the increase in running time which is equal to the turnaround time (i.e., the time between the job submission and successful completion) divided by the base job running time (with no waiting or premptions). Figure 6 compares the overhead (as percentage increase in running time) of SciSpot and HPC clusters for jobs of different lengths. We see that the average performance overhead due to waiting can be significant in the case of HPC clusters, and the job submission latency and queuing time dominate for

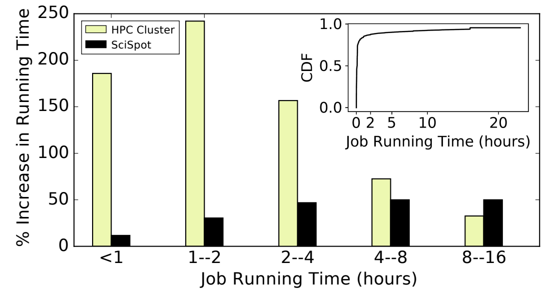

Figure 6: Increase in running time due to waiting/queuing on HPC clusters is significantly higher than the recomputation time for SciSpot, especially for shorter jobs.

# Application	Nodes	Big Red II	SciSpot
Nanoconfinement	1	2370	1546
Nanoconfinement	4	1140	851
Shapes	1	2649	1194
Shapes	4	1209	548

Table 1: Running times (in seconds) of different applications on the Big Red II HPC cluster vs SciSpot.

smaller jobs, increasing their total turnaround time by more 2.5×. This waiting is amortized in the case of longer running jobs, and the overhead drops off for longer jobs, to around 30%.

On the other hand, SciSpot's performance overhead is significantly smaller for jobs of up to 8 hours in length. For longer jobs, the limited lifetime of Google Preemptible VMs (24 hours) begins to significantly increase the preemption probability and expected recomputation time. We emphasize that these are *individual* job lengths, and not the running time of entire bag of jobs. We note that these large single jobs are rare, and for smaller jobs (within a much larger bag), both the preemption probability and recomputation overhead is much smaller. make more quantitative

Result: *While preemptions can increase running times due to recomputation, this increase is small, and is between 20 to 400% lower compared to the waiting times associated as overhead in conventional HPC clusters.*

6.1.4 Comparison with HPC Performance. The performance of scientific computing applications has been extensively compared on HPC and cloud setups [4, 12, 16, 35, 53]. For completeness, we show the running times on the Big Red II supercomputing cluster in Table 1, with 16 CPU nodes used throughout, and we see that our representative applications *do not* face a penalty when deployed on the cloud.

6.2 SciSpot Scaling

We now turn our attention to SciSpot's scaling properties. We are primarily interested in observing the behavior of running bags of jobs of different applications with different resource requirements. In all cases unless otherwise stated, we run bags of 36 jobs, and impose that 90% of all jobs complete (thus we target a completion of 32 jobs). The jobs in the

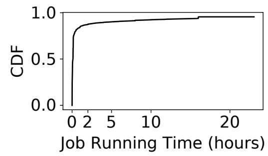

Figure 7: Most HPC jobs are less than 2 hours.

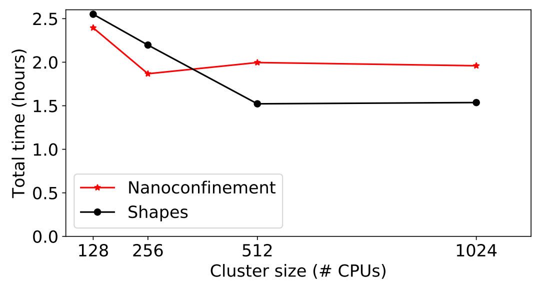

Figure 8: Bag of jobs running times exhibits classic parallel scaling behavior—performance improves until reaching a saturation point.

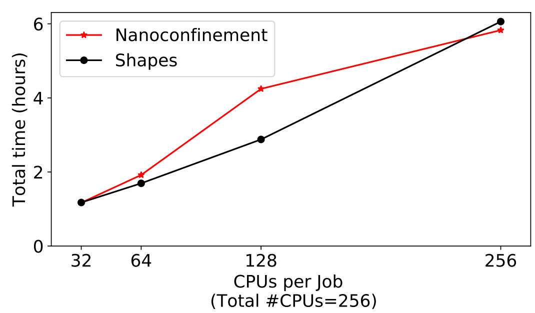

Figure 9: SciSpot can alleviate poor scaling by running more jobs in parallel and thus decreasing the intra-job parallelism (and hence number of CPUs per job, as shown in the figure).

bags are for exploring the different parameters (i.e., doing a parameter sweep), using SciSpot's automated parameter sweeping functionality described in Section ??. For reference, the distribution of running times for the different applications is shown in Figure 7. In the rest of this section, we evaluate SciSpot when the size of the cluster, the number of preemptions, and the number of jobs in the bag are increased.

6.2.1 Increasing Cluster Size. It is common to deploy scientific computing applications on large clusters, and we evaluate SciSpot on different cluster sizes in Figure 8. The figure shows the total running time (i.e., turnaround time) of the bag of jobs for the Nanoconfinement and Shapes applications as the total number of VMs (and hence total number of CPUs) increases. For this experiment, we used `n1-highcpu-32` VMs with 32 CPUs each, and we ran four jobs in parallel on the entire cluster. We see classic scaling behavior: both applications can scale to a higher number of VMs up to a point, after which communication overhead starts to dominate, and the performance saturates and we see no reduction in running time.

We note that SciSpot provides the option to alleviate the parallel scaling bottleneck, by increasing the number of parallel jobs. That is, for a given fixed cluster size, it can run more jobs in parallel, by reducing the total resources allocated and hence the parallelism of an individual job. The effect of changing the number of parallel jobs is shown in Figure 9, which shows the running time of an entire bag of jobs when the total cluster size is fixed (256 CPUs), but the number of parallel jobs and hence the number of CPUs per job changes. We see that a *smaller* number of CPUs per job limits the communication overhead, and thus reduces the total running time of the bag. For both the NC and Shapes application, we see up to 6× reduction in the total bag running time when more number of jobs are launched in parallel and a smaller number of CPUs per job are used.

Workload	Jobs	Time (Hours)	# Preemptions
Nanoconfinement	32	1.87	0
Nanoconfinement	100	6.08	1
Shapes	32	1.47	0
Shapes	100	4.49	5

Table 2: Running times and number of preemptions for bags of different sizes.

6.2.2 Increasing Bag Size. We now evaluate SciSpot's behavior when running larger bags of jobs. Table 2 shows the total running time of bags of 32 and 100 jobs. Since SciSpot reuses VMs when running jobs from a bag, it is able to take advantage of the relatively low preemption rates of VMs once they pass the first phase of early failures (Figure ??), and thus minimizes the number of preemptions as well as job failures. This makes SciSpot particularly suitable for running the large bags of jobs that are required when using machine learning techniques for HPC workloads, an emerging research area in many science and engineering disciplines [3, 6, 10, 11, 25, 26, 32, 33, 37, 47], since the training and testing data needed for statistical machine learning can be generated through SciSpot's bag of jobs execution model.

6.2.3 Increasing Preemptions. By reusing VMs across a bag of jobs and taking advantage of the low preemption rates during the middle of the 24 hour life of the preemptible VMs, the *expected* job failure rates and recomputation times are fairly small with SciSpot (as shown in Figures 4, 6). However, preemption rates can increase when the cloud operator sees high demand for resources. Figure 10 shows the running time of the bag of 32 Nanoconfinement jobs on a cluster of 4 `n1-highcpu-32` VMs, when different number of VMs are preempted.

We see that even with a high number of preemptions, the running time only increases by 50%. We note that a higher than expected preemption rate (as shown in the figure) is rare, and happens with a vanishingly small likelihood. This shows that SciSpot is robust and can provide acceptable performance even under extreme, adverse conditions.

7 RELATED WORK

SciSpot builds upon a large body of prior work on running scientific computing applications on the cloud, and the various facets of transient cloud computing.

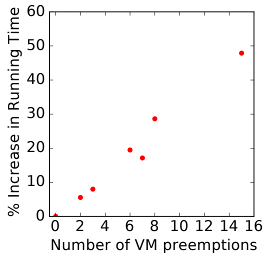

Figure 10: The increase in running time due to preemptions is under 50%, even when the number of preemptions is high.

7.1 Cloud Computing For Science

Running scientific applications on the cloud introduces many tradeoffs compared to conventional HPC clusters, along the dimensions of performance, cost, scalability, convenience, and reproducability. These tradeoffs are explored in [4, 12, 16, 35, 53]. In general, clouds can provide increased elasticity, lower waiting times, and more choices in resource allocation that can be tailored to the application. The cloud resource model is also present in platforms like nanoHUB [30], that provide easy execution and dissemination of nanotechnology simulation applications such as Ref. [27]. Outside of the bags of jobs execution model, price optimizations for scientific workflows in the cloud is discussed in [13]. While bags of tasks [46] are often used for parallel applications, SciSpot is the first to use of the bags of *jobs* abstraction for efficient and effective use of transient cloud servers.

7.2 Transient Cloud Computing

The challenges posed by Amazon EC2 spot instances, the first transient cloud servers, have received significant attention from both researchers and industry [**?**]. Since spot instances are significantly cheaper than the equivalent on-demand servers, they are attractive for running preemption and delay tolerant batch jobs [7, 21, 31, 43**? ? ?**]. A crucial component of EC2 spot instances is their dynamic auction-based pricing, and choosing the "right" bid price to minimize cost and performance degradation is the focus of much of past work on transient computing [15, 22, 36, 42, 45, 48–50, 52, 54, 55]. However, as explained in Section 3.1, it remains to be seen how Amazon's recent change [] in the preemption model of spot instances affects prior work.

On the other hand, the effective use of transient resources provided by other cloud providers such as Google, Microsoft, Packet, and Alibaba largely remains unexplored. Ours is the first work that studies the preemption characteristics and addresses the challenges involved in running large-scale applications on the Google Preemptible VMs, and provides insights on and leverages the unique preemption dynamics, as explained in Section 3.

7.2.1 Preemption Mitigation. Effective use of transient servers usually entails the use of fault-tolerance techniques such as checkpointing [38], migration [40], and replication. In the context of HPC workloads,[14, 34, 44] develop checkpointing and bidding strategies for MPI applications running on EC2 spot instances, based on spot pricing data and the older failure model. A comprehensive survey on periodic checkpointing for HPC applications can be found in [9].

By treating bags of jobs as an execution unit, allowing some jobs to fail, and using insights from preemption models, we show that it is possible to reduce the recomputation times to acceptable levels even without the use of periodic checkpointing that imposes additional deployment and performance overheads.

The first step towards mitigating preemptions is understanding their characteristics. Our preemption model for Google preemptible VMs developed in Section **??** extends the classic Weibull-distribution based bathtub models [**?**] by introducing exponential reclamation near the deadline and additional paramters that capture and explain the preemption dynamics.

7.2.2 Server Selection. Optimized server selection is an important problem in cloud computing, and especially for transient servers because of the cost-performance-preemption tradeoff involved. Similar to SciSpot, SpotOn [43] is also a batch computing service that selects servers based on job characteristics and failure rates of different EC2 spot VMs. However, it is restricted to individual, single-VM batch jobs, and its design is tied to EC2 spot instances. The state of the art transient server selection involves the use of multiple types of VMs [39], and selecting a heterogeneous cluster can reduce the likelihood of mass concurrent preemptions. However, since scientific computing applications are mostly synchronous, even a single failure affects the entire job, and heterogeneous clusters are not required, and are in fact, detrimental. Server selection is important even outside of preemptible VMs—developing bayesian optimization and application performance model based search for the "best" cloud VM is an active research area [1, 51].

8 CONCLUSION

Given the dramatic rise of transient cloud computing resources and their utilization in web services and distributed data processing, it is not the question of if, but when, transient cloud computing becomes a credible and powerful alternative to high-performance computing for scientific computing applications. In this paper, we developed principled approaches for deploying and orchestrating scientific computing applications on the cloud, and presented SciSpot, a framework for low-cost scientific computing on transient cloud servers. SciSpot develops the first empirical and analytical preemption model of Google Preemptible VMs, and uses the model for mitigating preemptions for "bags of jobs". SciSpot's cost-minimizing server selection and job scheduling policies can reduce costs by up to 5× compared to conventional cloud deployments. When compared to HPC clusters, SciSpot can reduce the total job turnaround times by more than 10×.

REFERENCES

[1] Alipourfard, O., and Yu, M. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. 15.

[2] Amvrosiadis, G., Park, J. W., Ganger, G. R., Gibson, G. A., Baseman, E., and DeBardeleben, N. On the diversity of cluster workloads and its impact on research results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)* (2018), pp. 533–546.

[3] Bartók, A. P., De, S., Poelking, C., Bernstein, N., Kermode, J. R., Csányi, G., and Ceriotti, M. Machine learning unifies the modeling of materials and molecules. *Science Advances 3*, 12 (2017).

[4] Benedictis, A. D., Rak, M., Turtur, M., and Villano, U. Cloud-Aware Development of Scientific Applications. In *2014 IEEE 23rd International WETICE Conference* (Parma, Italy, June 2014), IEEE, pp. 149–154.

[5] Bougeret, M., Casanova, H., Rabie, M., Robert, Y., and Vivien, F. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[6] Ch'ng, K., Carrasquilla, J., Melko, R. G., and Khatami, E. Machine learning phases of strongly correlated fermions. *Phys. Rev. X 7* (Aug 2017), 031038.

[7] Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., and Krintz, C. See Spot Run: Using Spot Instances for MapReduce Workflows. In *HotCloud* (June 2010).

[8] Daly, J. T. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems 22*, 3 (2006).

[9] Dongarra, J., Herault, T., and Robert, Y. Fault tolerance techniques for high-performance computing. 66.

[10] Ferguson, A. L. Machine learning and data science in soft materials engineering. *Journal of Physics: Condensed Matter 30*, 4 (2017), 043002.

[11] Fox, G., Glazier, J. A., Kadupitiya, J., Jadhao, V., Kim, M., Qiu, J., Sluka, J. P., Somogyi, E., Marathe, M., Adiga, A., et al. Learning everywhere: Pervasive machine learning for effective high-performance computation. *arXiv preprint arXiv:1902.10810* (2019).

[12] Galante, G., Erpen De Bona, L. C., Mury, A. R., Schulze, B., and da Rosa Righi, R. An Analysis of Public Clouds Elasticity in the Execution of Scientific Applications: a Survey. *Journal of Grid Computing 14*, 2 (June 2016), 193–216.

[13] Gar, Y., Monge, D. A., Mateos, C., and Garca Garino, C. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing* (Feb. 2019).

[14] Gong, Y., He, B., and Zhou, A. C. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (Austin, Texas, 2015), ACM Press, pp. 1–12.

[15] Guo, W., Chen, K., Wu, Y., and Zheng, W. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15* (Portland, Oregon, USA, 2015), ACM Press, pp. 191–202.

[16] Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems 22*, 6 (June 2011), 931–945.

[17] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. Simulation of charged systems in heterogeneous dielectric media via a true energy functional. *Phys. Rev. Lett. 109* (Nov 2012), 223905.

[18] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity. *The Journal of Chemical Physics 138*, 5 (2013), 054119.

[19] Jadhao, V., Thomas, C. K., and Olvera de la Cruz, M. Electrostatics-driven shape transitions in soft shells. *Proceedings of the National Academy of Sciences 111*, 35 (2014), 12673–12678.

[20] Jadhao, V., Yao, Z., Thomas, C. K., and De La Cruz, M. O. Coulomb energy of uniformly charged spheroidal shell systems. *Physical Review E 91*, 3 (2015), 032305.

[21] Jain, N., Menache, I., and Shamir, O. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *11th International Conference on Autonomic Computing (ICAC 14)*, USENIX Association.

[22] Javadi, B., Thulasiram, R., and Buyya, R. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *UCC* (December 2011).

[23] Jing, Y., Jadhao, V., Zwanikken, J. W., and Olvera de la Cruz, M. Ionic structure in liquids confined by dielectric interfaces. *The Journal of chemical physics 143*, 19 (2015), 194508.

[24] Joaquim, P., Bravo, M., Rodrigues, L., and Matos, M. Hourglass: Leveraging transient resources for time-constrained graph processing in the cloud. In *Proceedings of the Fourteenth EuroSys Conference 2019* (New York, NY, USA, 2019), EuroSys '19, ACM, pp. 35:1–35:16.

[25] Kadupitiya, J., Fox, G., and Jadhao, V. *Submitted* (2018).

[26] Kadupitiya, J., Fox, G., and Jadhao, V. Machine learning for performance enhancement of molecular dynamics simulations. Accepted.

[27] Kadupitiya, J., Marru, S., Fox, G. C., and Jadhao, V. Ions in nanoconfinement, Dec 2017. Online on nanoHUB; source code on GitHub at github.com/softmaterialslab/nanoconfinement-md.

[28] Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)* (Boston, USA, May 2013).

[29] Karlin, I., Keasler, J., and Neely, R. Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973, August 2013.

[30] Klimeck, G., McLennan, M., Lundstrom, M. S., and Adams III, G. B. nanohub. org-online simulation and more materials for semiconductors and nanoelectronics in education and research, 2008.

[31] Liu, H. Cutting MapReduce Cost with Spot Market. In *HotCloud* (June 2011).

[32] Liu, J., Qi, Y., Meng, Z. Y., and Fu, L. Self-learning monte carlo method. *Phys. Rev. B 95* (Jan 2017), 041101.

[33] Long, A. W., Zhang, J., Granick, S., and Ferguson, A. L. Machine learning assembly landscapes from particle tracking data. *Soft Matter 11*, 41 (2015), 8141–8153.

[34] Marathe, A., Harris, R., Lowenthal, D., De Supinski, B. R., Rountree, B., and Schulz, M. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC* (2014), ACM.

[35] Marathe, A., Harris, R., Lowenthal, D. K., De Supinski, B. R., Rountree, B., Schulz, M., and Yuan, X. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing* (2013), ACM, pp. 239–250.

[36] Mihailescu, M., and Teo, Y. M. The Impact of User Rationality in Federated Clouds. In *CCGrid* (2012).

[37] Schoenholz, S. S. Combining machine learning and physics to understand glassy systems. *Journal of Physics: Conference Series 1036*, 1 (2018), 012021.

[38] Sharma, P., Guo, T., He, X., Irwin, D., and Shenoy, P. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys* (April 2016).

[39] Sharma, P., Irwin, D., and Shenoy, P. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems* (June 2017), vol. 1, p. 23.

[40] Sharma, P., Lee, S., Guo, T., Irwin, D., and Shenoy, P. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys* (April 2015).

[41] Solis, F. J., Jadhao, V., and De La Cruz, M. O. Generating true minima in constrained variational formulations via modified lagrange multipliers. *Physical Review E 88*, 5 (2013), 053306.

[42] Song, Y., Zafer, M., and Lee, K. Optimal Bidding in Spot Instance Market. In *Infocom* (March 2012).

[43] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[44] Taifi, M., Shi, J. Y., and Khreishah, A. SpotMPI: A Framework for Auction-Based HPC Computing Using Amazon Spot Instances. In *Algorithms and Architectures for Parallel Processing*, Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7017. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 109–120.

[45] Tang, S., Yuan, J., and Li, X. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. In *CLOUD* (June 2012).

[46] Varshney, P., and Simmhan, Y. AutoBoT : Resilient and Cost-effective Scheduling of a Bag of Tasks on Spot VMs. *IEEE Transactions on Parallel and Distributed Systems* (2019), 1–1.

[47] Ward, L., Dunn, A., Faghaninia, A., Zimmermann, N. E., Bajaj, S., Wang, Q., Montoya, J., Chen, J., Bystrom, K., Dylla, M., et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science 152* (2018), 60–69.

[48] Wee, S. Debunking Real-Time Pricing in Cloud Computing. In *CCGrid* (May 2011).

[49] Wolski, R., Brevik, J., Chard, R., and Chard, K. Probabilistic guarantees of execution duration for Amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17* (Denver, Colorado, 2017), ACM Press, pp. 1–11.

[50] Xu, H., and Li, B. A Study of Pricing for Cloud Resources. *Performance Evaluation Review 40*, 4 (March 2013).

[51] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17* (Santa Clara, California, 2017), ACM Press, pp. 452–465.

[52] Zafer, M., Song, Y., and Lee, K. Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs. In *CLOUD* (2012).

[53] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[54] Zhang, Q., Gürses, E., Boutaba, R., and Xiao, J. Dynamic Resource Allocation for Spot Markets in Clouds. In *Hot-ICE* (March 2011).

[55] Zheng, L., Joe-Wong, C., Tan, C. W., Chiang, M., and Wang, X. How to Bid the Cloud. In *SIGCOMM* (August 2015).